INSERTION SORT and BINARY SEARCH

April 5, 2017

1 INSERTION SORT

1.1 Introduzione

L' Insertion Sort é un algoritmo relativamete semplice per ordinare un array e molto efficente per ordinare un piccolo numero di elementi. É un algoritmo chiamato "in place", cioé ordina l'array senza doverne creare un altro di appoggio, quindi risparmia memoria.

Non é molto diverso dal modo in cui un essere umano, spesso, ordina un mazzo di carte nella propria mano.

1.2 Ordinare un array numerico

La sequenza é logicamente suddivisa in due parti: una ordinata ed una non ordinata (all'inizio la sottosequenza ordinata sará formata solamente dal primo valore)

Viene preso un valore alla volta dalla sottosequenza non ordinata e si inserisce nella sottosequenza ordinata

Per trovare la giusta posizione del valore appena prelevato dalla sottosequenza non ordinata, confrontiamo il valore con gli altri presenti in quella ordinata, da destra verso sinistra. Ogni valore piú grande verrá spostato verso destra in modo da fare posto al valore da inserire.

1.3 Algoritmo

```
In [31]: A = [21,100,9,43,33,73,19,93,1,80]
```

```
In [32]: for j in range(1,10):
             key = A[j]
             i=j-1
             while (i>=0 and A[i]>key):
                 A[i+1]=A[i]
                 i-=1
             A[i+1]=key
             print A
```

```
[21, 100, 9, 43, 33, 73, 19, 93, 1, 80]
[9, 21, 100, 43, 33, 73, 19, 93, 1, 80]
[9, 21, 43, 100, 33, 73, 19, 93, 1, 80]
[9, 21, 33, 43, 100, 73, 19, 93, 1, 80]
[9, 21, 33, 43, 73, 100, 19, 93, 1, 80]
[9, 19, 21, 33, 43, 73, 100, 93, 1, 80]
[9, 19, 21, 33, 43, 73, 93, 100, 1, 80]
[1, 9, 19, 21, 33, 43, 73, 93, 100, 80]
[1, 9, 19, 21, 33, 43, 73, 80, 93, 100]
```

2 BINARY SEARCH

2.1 Introduzione

L'algoritmo Binary Search (Ricerca Binaria o Dicotomica) è simile al metodo usato per trovare una parola sul dizionario: sapendo che il vocabolario è ordinato alfabeticamente, l'idea è quella di iniziare la ricerca non dal primo elemento, ma da quello centrale, cioè a metà del dizionario. Si confronta questo elemento con quello cercato:

se corrisponde, la ricerca termina indicando che l'elemento è stato trovato;

se è superiore, la ricerca viene ripetuta sugli elementi precedenti (ovvero sulla prima metà del dizionario), scartando quelli successivi;

se invece è inferiore, la ricerca viene ripetuta sugli elementi successivi (ovvero sulla seconda metà del dizionario), scartando quelli precedenti.

se si arriva al punto che tutti gli elementi vengono scartati, la ricerca termina indicando che il valore non è stato trovato.

L'algoritmo trova applicazione anche per realizzare un programma che indovina un numero naturale (casuale o scelto dall'utente) compreso in un intervallo. Infatti si può ricondurre tale situazione alla ricerca di un elemento su un insieme ordinato che ha, come dati, tutti i numeri naturali compresi nell'intervallo, ed è proprio il nostro caso.

2.2 Algoritmo

Nota: L'algoritmo è stato sviluppato mediante un implementazione ricorsiva.

```
In [33]: def BinarySearch(A,i,n,v):
             if i>n or v<A[i] or v>A[n-1]:
                 return -1
             m = (n+i) / 2
             if A[m] == v:
                 return m
             elif A[m] > v:
                 return BinarySearch(A,i,m,v)
             else:
                 return BinarySearch(A,m+1,n,v)
```

Ricerca del valore 73

```
In [34]: v = 73

         pos = BinarySearch(A,0,len(A),v)
         if(pos==-1):
             print "Valore non trovato"
         else:
             print "Valore trovato in posizione "+str(pos)
```

```
Valore trovato in posizione 6
```